structuration d'un systeme téléphonique informatisé

*m.exel
m.mekinda
b.popovič

*Institut J.Stefan, Jamova 39, 61000 Ljubljana
Iskra, ATC Labore, Savska Loka 4, 64000 Kranj

UDK 621.395.345:681.3.06

Dans le présent article nous appliquons les principes de la programmation structurée et (pseudo)parallèle pour analyser et restructurer - au moyen d'un langage de haut niveau utilisant le concept de moniteur - un système téléphonique informatisé concret.

Članek opisuje aplikacijo principov strukturiranega in (navidezno) paralelnega programiranja. Predmet aplikacije je analiza in strukturiranje konkretnega programskega sistema telefonske centrale s pomočjo visoko nivojskega programskega jezika, ki uporablja monitorski koncept.

I. INTRODUCTION

Nous nous proposons ici d'analyser et de restructurer un système téléphonique informatisé concret en nous appuyant sur les concepts de programmation structurée et de programmation (pseudo-)parallèle. Ceci n'est pas une étude théorique; nous essayons simplement d'examiner l'applicabilité des concepts mentionnés à un système en temps réel: le systeme téléphonique considéré. La parution récente de quelques articles (5,14) traitant de la programmation en temps réel en liaison avec les méthodes de structuration des systèmes d'exploitation et avec les langages pour la programmation parallèle nous a encouragé à procéder de la sorte.

Les motivations de cette étude sont pratiques: il s'agit de la maintenance, de l'adaptation et des modifications du système informatisé des centraux téléphoniques du type Metaconta 10C Local, réalisé avec les ordinateurs ITT-1600 et 3200. Le logiciel du système est programmé en assembleur et toute modification est difficile. Ceci nous a amené à étudier une restructuration possible du systeme qui permette:
- des modifications aisées et facilement vérifiables;
- une compréhension meilleure du système et, éventuellement,
- une amélioration du degré de parallélisme (potentiel) des différents éléments du système.

Nous présentons dans les chapitres suivants une analyse et une structuration de ce système téléphonique en termes de *processus coopérants, une description de cette structuration* dans un langage qui est pratiquement le Pascal parallèle (2,16) et, finalement, nous discutons des problèmes que pose l'organisation de cette structuration.

II. PRÉSENTATION ET ANALYSE DU SYSTEME

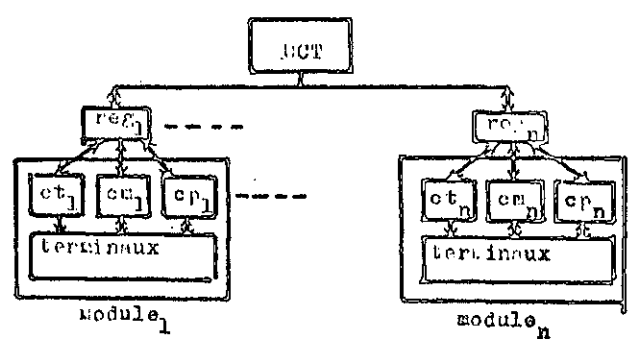
1. Présentation schématique du système

Le système du central téléphonique considéré comprend le matériel suivant: un ou deux ordinateurs ITT-1600 et une périphérie téléphonique comprenant des terminaux d'entrées-sorties, des terminaux de liaison, des éléments de liaison entre ces terminaux et des circuits de tests, de marquage et de pilotage dirigeant ces terminaux. Les terminaux d'E/S sont reliés par des canaux à l'environnement que composent des appareils téléphoniques, d'autres centraux etc...

L'organisation physique de la périphérie téléphonique est modulaire: cette périphérie se compose d'un module de signalisation et au plus de dix modules périphériques. Un module périphérique est composé d'un certain nombre de terminaux (d'E/S et de liaison) et d'éléments de liaison que dirigent un circuit de marquage, trois circuits de tests de types différents et jusqu'à trois circuits de pilotage de types lent, normal et rapide. Nous ne tiendrons pas compte dans la suite de la périphérie classique des ordinateurs.

Fonctionnellement nous distinguons dans ce système du central deux sous-systèmes: le système de contrôle réalisé sur les ordinateurs et le système de coopération avec l'environnement réalisé par la périphérie téléphonique. Le système de contrôle contrôle le système de coopération et traite les messages que ce dernier lui communique alors que le système de coopération détecte les signaux provenant de l'environnement, envoie des signaux à ce dernier et établit des liaisons entre les canaux connectés à l'environnement.

Dans la suite nous considérerons un système simplifié comprenant une seule unité centrale de traitement et n modules périphériques dont chacun, relié à l'UCT par un registre périphérique, contient un circuit de tests, un circuit de marquage et un circuit de pilotage de type normal. Un schéma partiel de ce système est présenté dans la figure 1.



ct: circuit de teste
cm: circuit de marquage
cp: circuit de pilotage normal

Figure 1.

2. Processus du systeme: identification et coopération

Nous analysons ici les deux systèmes de contrôle et de coopération en termes de processeurs et de processus séquentiels permanents et coopérants.

Le système du central comprend un processeur principal (l'UCT), un processeur d'horloge à période de 10 msec et les trois fois n processeurs périphériques de tests, de marquage et de pilotage qui sont constitués des circuits correspondants des n modules périphériques et des terminaux de la périphérie téléphonique multiplexés sur ces circuits.

Les processus du système de contrôle se déroulent sur le processeur principal et sur le processeur d'horloge et sont appelés processus internes alors que les processus du système de coopération se déroulent sur les processeurs périphériques de tests, de marquage et de pilotage et sont appelés processus externes (voir la terminologie de (1)) de même nom, respectivement.

Les processus externes de tests réalisent la détection - par l'examen des points de tests des terminaux - des signaux parvenant de l'environnement; les processus externes de pilotage réalisent l'envoi des signaux à l'environnement par l'établissement des liaisons entre les terminaux et les canaux de l'environnement; enfin, les processus externes de marquage réalisent des liaisons entre les terminaux. Il y a quatre groupes de processus internes: les processus internes de tests, de traitement, de marquage et de pilotage, dénotés de façon générique par pt, ptr, pm et pp, respectivement.

Les fonctions de ces processus sont les suivantes: un processus pt détermine un ensemble de points de tests et active un processus externe de tests sur un processeur périphérique donné en lui envoyant une commande par le registre périphérique associé au processeur. Le processus externe prépare sa "réponse" dans le meme registre périphérique qui doit rester occupé par ce processus pendant son activité. Le processus pt analyse ensuite la réponse, envoie un message par l'intermédiaire d'une mémoire-tampon à un processus ptr couplé a pt et reprend la procédure au début jusqu'à ce que l'ensemble de points de tests soit traité. Les processus internes de tests sont activés périodiquement par l'interruption de l'horloge et les différents processus pt se déroulent alors séquentiellement.

Un processus ptr analyse les messages reçus de pt et envoie, en fonction du message reçu, un message à un processus pm ou pp par l'intermediaire d'une mémoire-tampon ce qui entraîne une interruption activant pp ou pm.

Un processus pp ou pm reçoit des messages des processus ptr et active un processus externe de pilotage ou de marquage, en lui envoyant une commande par le registre périphérique. Le processus externe activé occupe le registre tant que la commande n'est par lue et indique la fin de son activité par une interruption ce qui permet de continuer le processus interne qui l'a activé. Le processus externe n'envoie pas d'autre "réponse".

Nous voyons donc qu'il existe une communication entre les processus internes et externes et entre les différents processus internes. Cette communication se fait par les registres périphériques et les mémoires tampons; ces éléments représentent des ressources partagées que les processus devrons allouer et déallouer dans notre description du système. D'autres ressources partagées sont les processeurs périphériques qui devront être alloués avant l'activation des processus externes se déroulant sur ces processeurs.

L'occupation exclusive des ressources partagées est assurée dans le système actuel par l'exécution sérielle des processus internes. La communication entre les processus du système actuel n'est ni sûre (des messages peuvent se perdre) ni suffisamment souple pour permettre d'exploiter le parallélisme potentiel des processus du système. Nous proposons donc une restructuration du système exposée ci-dessous.

3. Structuration du système

Le système est envisagé comme un ensemble de processus internes séquentiels et permanents (cyclant indéfiniment) dont la coopération est assurée par un ensemble de moniteurs (pour le concept de moniteur voir 1,2,3,4,7,11,12, 13,16), disposés en hierarchie et gérant les ressources du système.

Le noyau du système n'est pas explicité; nous supposons qu'il contient essentiellement un programme d'ordonnancement ("scheduler - dispatcher") des processus internes, les routines pour le traitement des interruptions et deux primitives de communication entre les processus: "bloquer" et "débloquer". Ces deux primitives sont appelées dans les moniteurs et appellent à leur tour le programme d'ordonnancement. Nous supposons un ordonnancement tres simple des processus internes: absence de priorités et stratégie "premier arrivé - premier servi". Notons que la procédure "débloquer" ne provoque pas nécessairement l'activation immédiate du processus débloqué éventuel (nous ne suivons donc pas la stratégie de Hoare et de Brinch Hansen - cf. 2,7 - mais celle, plus générale, exposée dans 18).

Dans le système observé n'existent que deux types d'interruptions: l'interruption de l'horloge et les interruptions provoquées par les processeurs de marquage et de pilotage. Ces interruptions sont "interceptées" par le noyau (qui active par la suite le programme d'ordonnancement) et finalement "acheminées" aux procédures adéquates des moniteurs gérant les processeurs de marquage, de pilotage et l'horloge.

L'occupation exclusive des ressources partagées est assurée par l'inhibition des interruptions dans les procédures des moniteurs ce qui a pour effet l'exclusion "globale" (cf. 11) de ces procédures.

Le système proposé se compose d'éléments suivants:

- processus internes:
- les processus de tests, notés pt_i, $i = 1..t$
- les processus de traitement, notés ptr_i, $i = 1..t$
- les processus de marquage, notés pm_i, $i = 1..m$
- les processus de pilotage, notés pp_i, $i = 1..p$;
- moniteurs:
- les moniteurs des processeurs de tests, notés mt_i, $i = 1..n$
- les moniteurs des processeurs de marquage, notés mm_i, $i = 1..n$
- les moniteurs des processeurs de pilotage, notés mp_i, $i=1..n$
- les moniteurs des registres périphériques, notés mr_i, $i=1..n$
- les moniteurs notés $mb1_i$, $i=1..t$, gérant la communication entre les paires des processus pt_i - ptr_i
- les moniteurs notés $mb2_i$, $i=1..m$, gérant la communication entre les processus ptr et les processus pm_i
- les moniteurs notés $mb3_i$, $i=1..p$, gérant la communication entre les processus ptr et les processus pp_i et
- le moniteur noté mtemp gérant l'interruption de l'horloge.

L'organisation du système est représentée par le graphe de la figure 2. Sur ce graphe apparaissent aussi le processus d'horloge noté ph et les processus externes de marquage et de pilotage, notés cp_i et cm_i, $i=1..n$. Les arcs du graphe représentent les accès (appels) admis dans le système (une interruption étant ici interprétée comme un appel à la procédure d'interruption - de moniteur - correspondante). L'arc partant d'une accolade (resp. aboutissant à une accolade) représente autant d'arcs partants (resp. aboutissants) qu'il y a de processus ou de moniteurs réunis par l'accolade.

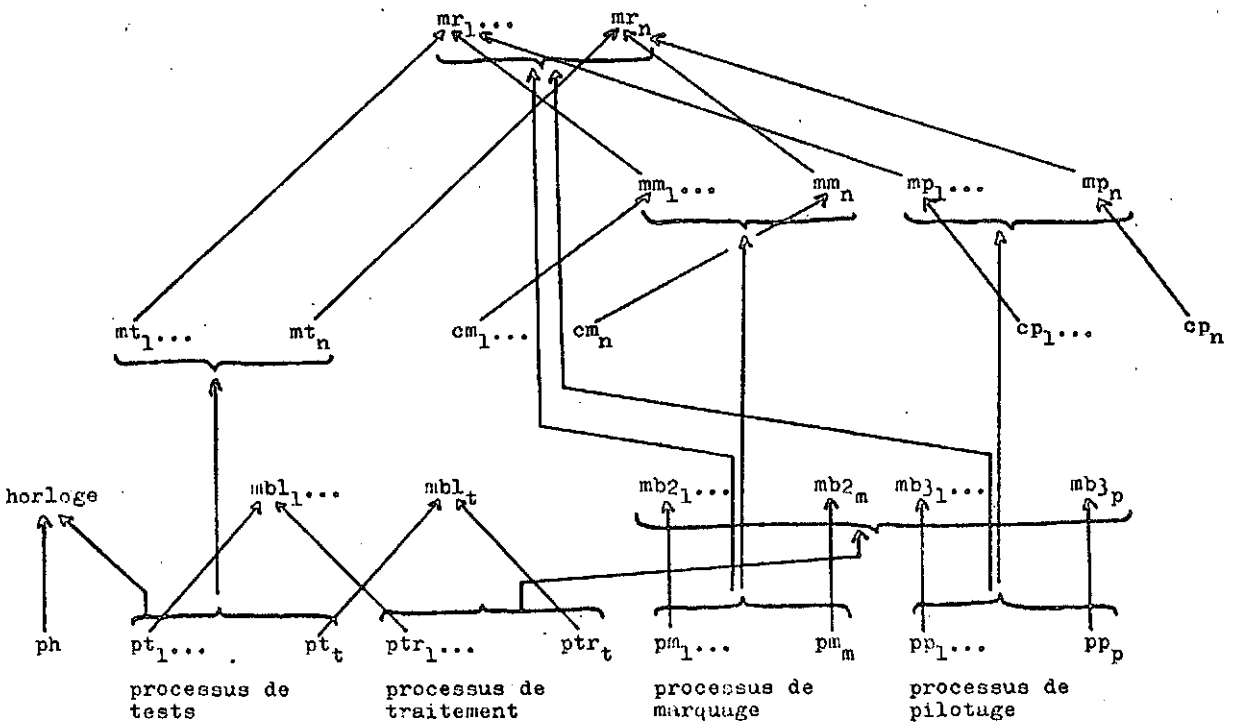


Figure 2.

III. DESCRIPTION DE LA STRUCTURATION PROPOSÉE

Différentes structurations du système sont possibles; elles sont essentiellement fonction du choix des moniteurs. La structure choisie a les effets suivants:

- en allouant d'abord les processeurs de tests, de marquage et de pilotage et ensuite les registres correspondants nous obtenons des files d'attende (sur ces ressources) plus courtes que dans le cas des séquences d'allocation inverses;
- il est possible que les processus externes de marquage et de pilotage d'un même module périphérique se déroulent en parallèle;
- en définissant un moniteur pour chaque ressource nous diminuons le temps passé dans chacune des procédures des moniteurs.

1. Données globales

Ces données globales peuvent etre référencées dans tous les moniteurs et dans tous les processus.
Les constantes: t,m,p,n,blon1,blon2,blon3.
Les types:

```
type liste-t = array[1..t] of queue;
     liste-m = array[1..m] of queue;
     liste-p = array[1..p] of queue;
     liste-r = array[1..2] of queue;
     mess1 ={ ...}; mess2 ={...}; mess3 ={ ...};
     tamp1 = array[1..blon1] of mess1;
     tamp2 = array[1..blon2] of mess2;
     tamp3 = array[1..blon3] of mess3;
type paps = class (long: integer); {cf.(15), p.169: une file
                        "premier arrivé-premier servi"}
     var tête,q,l: integer;
     function entry arrivé: integer;
     begin arrivé:=q; q:=q mod long + 1; l:=l + 1; end;
     function entry départ: integer;
     begin départ:=tête, tête:=tête mod long + 1; l:=l - 1;end;
     function entry vide: boolean;
     begin vide:=(l=0) end;
     function entry plein: boolean;
     begin plein:=(l=long) end;
     begin tête:=1; q:=1; l:=0; end;
```

Les primitives du noyau:

```
procedure bloquer (var q:queue);
  {bloquer le processus courant dans q; quitter le moniteur
   et appeler le programme d'ordonnancement};
procedure débloquer (var q: queue);
  {débloquer le processus dans q; quitter le moniteur et
   appeler le programme d'ordonnancement};
function vide (var q:queue):boolean;
  {vrai si pas de processus en attente dans q}.
```

2. Moniteurs

a) Allocateur de processeurs de tests

```
type mt=
monitor(monreg: mr)
var libre: boolean; liste: liste-t; prochain: paps;
procedure entry allouer;
begin while not libre do bloquer (liste[prochain.arrivé]);
      libre:=false;
      monreg.allouer;
end;
procedure entry déallouer;
begin monreg.déallouer;
      libre:=true;
   if not prochain.vide then débloquer(liste[prochain.départ]);
end;
begin libre:=true; init prochain(t) end;
```

On peut declarer alors n moniteurs du type mt dont chacun est associé à un moniteur du type mr (voir plus bas):
`var mt1:mt; avec init mt1(mr1);`

b) Allocateurs de processeurs de marquage et de pilotage

```
type mm=
monitor(monreg: mr)
var libre:boolean; liste:liste-m; prochain: paps;
    inter:boolean; attinter:queue;
procedure entry allouer;
```

```
begin while not libre do bloquer(liste[prochain.arrive]);
      libre:=false;
      monreg.allouer;
      inter:=false; { le signal d'interruption }
end;
procedure entry déallouer;
begin if not inter then bloquer(attinter);
     {attendre l'interruption }
      libre:=true;
      if not prochain.vide then débloquer(liste[prochain.départ]);
end;
procedure entry minter;{"intercepte" l'interruption du circuit
                        de marquage }
begin inter:=true;
      if not vide(attinter) then débloquer(attinter);
end;
begin libre:=true; inter:=false; init prochain(m) end;
type mp= { analogue au type mm }.
 On pourra déclarer n moniteurs du type mm et n moniteurs
 du type
mp par:
 var mm1:mm;  avec  init mm1(mr1);  etc...
```

c) Allocateur de registre périphérique

```
type mr=
monitor
var libre:boolean; liste:liste-r; prochain:paps;
procedure entry allouer;.
begin while not libre do bloquer(liste[prochain.arrivé]);
      libre:=false;
end;
procedure entry déallouer;
begin libre:=true;
      if not prochain.vide then debloquer
      (liste[prochain.départ]);
end;
 On déclarera n moniteurs du type mr par:
var mr1:mr;  etc...
```

d) Allocateur de mémoire-tampon du type mb1

```
type mb1=
monitor
var mb:tamp1; prochain:paps; source, destination:queue;
procedure entry envoyer(m:mess1);
begin if prochain.plein then bloquer(source);
      mb[prochain.arrivé]:=m;
       if not vide(destination) then débloquer(destination);
end;
procedure entry recevoir(var m:mess1);
begin if prochain.vide then bloquer(destination);
      m:= mb[prochain.départ];
      if not vide(source) then debloquer(source);
end;
begin init prochain(blon1) end;
 On déclarera t moniteurs de type mb1: var mb11:mb1; etc...
```

e) Allocateurs des mémoires-tampons du type mb2 et mb3

```
type mb2=
monitor
var mb:tamp2; prochainproc,prochaintamp:paps;
    source:liste-t; destination:queue;
procedure entry envoyer(m:mess2);
begin while prochaintamp.plein do bloquer
      (source[prochainproc.arrivé]);
      mb[prochaintamp.arrivé]:=m;
      if not vide(destination) then débloquer(destination);
end;
procedure entry recevoir(var m:mess2);
begin if prochaintamp.vide then bloquer(destination);
      m:=mb[prochaintamp.départ];
      if not prochainproc.vide then
        debloquer(source[prochainproc.départ]);
end;
begin init prochainproc(t), prochaintamp(blon2)  end;
type mb3 = { analogue au type mb2 }.
```

On déclarera m moniteurs du type mb2 et p moniteurs du type mb3:

```
var mb21:mb2,...   mb3l:mb3,...
```

f) Le moniteur de l'horloge

```
type mtemp=
monitor
var attliste:liste-t; {contient les processus pt; qui ont fini leurs
    cycles et attendent l'interruption de l'horloge pour
    recommencer un nouveau cycle}
    prochain:pops;
procedure entry attendre;
begin bloquer(attliste[prochain.arrivé])  end;
procedure entry tempinter;{l'interruption de l'horloge}
begin while not prochain.vide
      do débloquer(attliste[prochain.départ])
end;
begin init prochain(t)  end;
 Déclaration:  var horloge:mtemp;
```

3. Processus

a) Processus de tests

```
type pt=
process(heure:mtemp; buf:mb1; accèst1,{...}, accèstn:mt);
var circuitest:1...n; mess:mess1;
begin
   cycle
   {initialisation:déterminer les tests}
     repeat
     ...
     case circuitest of 1:accèst1.allouer;
    {envoi de la commande au processeur de tests
     correspondant en utilisant le registre correspondant;
     delai "actif"△; examen de la reponse reçue dans le
     registre}
     case circuitest of 1:accèst1.déallouer;
    {preparer le message pour le processus ptr correspondant}
     buf.envoyer(mess);
     until {fin de tests du cycle};
   heure.attendre;
   end;
end;
 Les processus de type pt sont déclarés et initialisés par:
var pt1:pt; init pt1(horloge,mb11,mt1,mt2,{...},mtn); etc..
```

b) Processus de traitement

```
type ptr=
process(buf:mb1; tamp1,{...},tampm:mb2 ; tam1,{...},
        tamp:mb3);
var m1:mess1; m2:mess2; m3:mess3;
begin
   cycle   buf.recevoir(m1);
          {traitement du message}
      ...{tamp1.envoyer(m2) ... tam1.envoyer(m3)}...
   end;
end;
 Ces processus sont déclarés et initialisés par:
var ptr1:ptr; init ptr1(mb11,mb21,{...},mb31,{...},mb3p);
```

c) Processus de pilotage et de marquage

```
type pm=
process(buf:mb2; accèsm1,{...}, accèsmn:mn;
                 accèsr1,{...}, accèsrn:mr);
var circuitmarq:1..n; mess:mess2;
begin
 {initialisation}
 cycle buf.recevoir(mess);
  {préparer l'action adéquate}
  case circuitmarq of 1:accèsm1.allouer;
                      ...
   {envoi de la commande au processeur de marquage
    correspondant en utilisant le registre correspondant}
  case circuitmarq of 1:begin accesr1.deallouer;
                              accesm1.déallouer;
                      end;
                      ...
end;  end;
```

Ces processus sont déclarés et initialisés par:
var pm1:pm;
init pm1(mb21,mm1,{...},mmn,mr1,mr2,{...},mrn); etc...
type pp= { analogue au type pm}.

IV. CONCLUSION

Le fonctionnement correct du système décrit en III (ainsi que celui du système original) est basé sur l'hypothèse suivante: *l'intervalle de temps entre deux interruptions de* l'horloge est sufisamment long pour que, en fin d'intervalle:
- tous les processus pt soient bloqués dans la liste "attliste",
- aucun des messages ne soit perdu et
- *tous les processus externes soient terminés.*

Partant de cette hypothèse de base nous pouvons supposer que le système structuré proposé ici assure un déroulement harmonieux des processus du système c.à.d. sans blocuage des processus (cf. 1).
En effet, d'une part, nous avons dans le système une allocation hiérarchique des ressources permanentes (processeurs périphériques et registres) et, d'autre part, une communication de messages hiérarchique entre les processus internes du système (les processus pt envoient des messages aux processus ptr et ces derniers envoient des messages aux processus pm et pp).

L'implantation du système proposé est basée sur l'hypothèse d'un seul processeur central et est la plus simple possible (les moniteurs sont réalisés par l'inhibition des interruptions). On peut bien évidemment envisager une implantation plus sophistiquée et plus complexe (cf. 17):
- en supposant plusieurs processeurs centraux et/ou
- en introduisant les sémaphores pour assurer une exclusion "locale" des procédures des moniteurs.

Une telle implantation rendrait possible une amélioration théorique du degré de parallélisme des différents processus du système mais - étant donné les temps très courts des cycles des processus - serait probablement prohibitive relativement aux temps d'"accès" aux moniteurs.

Nous pensons que la restructuration proposée du système du central peut, d'une part, servir à modéliser le système et à le simuler et, d'autre part, peut servir comme point de départ d'une configuration nouvelle du système.

BIBLIOGRAPHIE

(1) Brinch Hansen P.: Operating system principles, Prentice-Hall, 1973.
(2) Brinch Hansen P.: The programming language Concurrent Pascal, IEEE trans. on software eng. vol. SE-1, no.2, 1975.
(3) Brinch Hansen P.: A programming methodology for operating system design, IFIP 1974.
(4) Dijkstra E.W.: Hierarchical ordering of sequential processes, Operating systems techniques, Academic Press, 1972.
(5) Gordon R.L.: Systems of cooperating schedulers, IFAC-IFIP workshop on real-time programming, 1975.
(6) Haberman A.N.: Introduction to operating system design, SRA 1976.
(7) Hoare C.A.R.: *Monitors: an operating system structuring* concept, CACM, oct. 1974, p. 549.
(8) Horning J.J., Randell B.: Process structuring, Computing surveys, vol. 5, no.1, 1973.
(9) ITT: *Standard computer modules* ITT 1600, 160 ITT 11000E, 1968.
(10) Jensen K., Wirth N.: Pascal-user manual and report, Springer-Verlag, 1975.
(11) Lister A.M., Maynard K.J.: An implementation of monitors, Software-practice & experience, vol. 6, 377-385, 1976.
(12) Lister A.M., Sayer P.J.: Hierarchical monitors, Proc. 1976 internat. conf. on parallel processing.
(13) Schmid H.A.: On the efficient implementation of conditional critical regions & the construction of monitors, Acta Informatica 6, 1976.
(14) Smedema C.H.: Real-time concepts and Concurrent Pascal, IFAC-IFIP workshop on real-time programming, 1975.
(15) Brinch Hansen P.: The solo operating system: processes, monitors & classes, Software-practice & experience, vol.6, 165-200, 1976.
(16) Brinch Hansen P.: Concurrent Pascal report, Cal.tech., 1975.
(17) Wettstein H.: The implementation of synchronizing operations in various environments, Software-practice & experience, vol.7, 115-126, 1977.
(18) Wettstein H.: The problem of nested monitor calls revisited, Oper. Systems Review, vol.12, no.1, 1978.
(19) Popović B., Exel M., Mekinda M.: Primjena monitorskog koncepta u izgradnji operacionog sistema za periodično aktiviranje programa. J. Informatica, 1977, no. 2.
(20) Mekinda M., Exel M., Popović B.: Strukturiranje programsko vodjenog sistema telefonske centrale, XII.jug. medn. simpozij o obravnavanju podatkov, Bled, oktober 1977, 6-116.